University of Puerto Rico
Mayagüez, Puerto Rico
Electrical and Computer Engineering Department

MOSIS Research Final Report

Date: January 26, 2023

By:
William Class

For:
Dr. Manuel Jimenez

Abstract

The primary purpose of this poster is to inform the progress made on the completion of the project during the first semester of the academic year 2022-2023. The overall project, proposed by David Repollet, consists in developing a Marine Operated Sensing Imaging Stereoscope (M.O.S.I.S). Work has been primarily focused in the sensor hub, which began during the month of August of this academic year. This semester was focused in finishing and fully implementing the sensor hub. As of this point, communication with the four sensors has been successfully established, MCU has been properly configured and initialized, sleep functionality has been configured, PCB board has been designed, system diagnostic and error-handling routines have been developed, sensors have been calibrated, and a means for data extraction has been developed .The overall project objective is to determine effectiveness of MOSIS prototype vs other conventional methods. In order to accomplish this, the effectiveness of this prototype will be evaluated by applying a study of diseases using two methods: In-situ observations and sample extraction, to be observed in a laboratory. Observations of diseases such as SCTLD, Coral Bleaching, and other coralline border phenomena.

Table of Contents

List of figures

1 Introduction

Conducting studies and microscopic observations on aquatic organisms is no easy task, there are a lot of conditions and details that must be taken into account in order to yield accurate results. Conditions and details that, as peculiar and distinct as they may be, must be artificially recreated in a lab to conduct a proper study. A proper lab may be equipped with a wide variety of tools, but few of these tools work in an underwater environment. Furthermore, once the organism is removed from its natural habitat, they lose some of their in-situ characteristics; organisms such as: small sponges, corals, and ascidians. In order to mitigate this problem, the Marine Operated Sensing Imaging Stereoscope (M.O.S.I.S) was proposed by David Repollet. M.O.S.I.S allows for in-site microscopic and stereoscopic image capturing, as well as environmental data acquisition, such as: temperature, pH, DO, and pressure. Ultimately, M.O.S.I.S acts as an in-site lab, removing the need to transport the collected sample onto a laboratory for further study.

This instrument will allow observing in-situ the progression of diseases such as SCTLD, Coral Bleaching, and other coralline border phenomena. The development and completion of MOSIS will allow to capture underwater stereoscopic/microscopic images with a log of fundamental sensor readings from the surrounding environment. MOSIS is equipped with commercial, off-the shelf thermometer, barometer, pH, and dissolved oxygen sensors. The instrument will be manipulated via SCUBA diving. For is use, a researcher places the device over or looking towards the selected sample area and selects a three-lighting color mode under which to capture the image in near infra-red (NIR), white, and near ultraviolet (NUV) light. From here the user must be able to achieve successfully focused still images, timelapse images, a customizable image stacking sequences, record video and recover ambient sensor data from the marine environment in-situ which can be studied at a future occasion at the laboratory via a 3D headset and post-mortem image analysis of the meta-data recovered. The integration of sensors into the microscope is aimed to reduce the percentage of human error in experiments conducted in the marine environment where portable means of imaging and frequent water sampling are needed. By automating part of the sampling procedure and sampling sensory data along with image data, the imaging device provides a stable means of observing and correlating small sessile benthic organisms in their natural environment without removing them and altering experimental results.

2. System configuration

2.1 System Overview

The device can be divided into two modules, the optical module and the sensor hub. This report will give emphasis on the sensor hub. The sensor hub consists of four distinct sensors; sensors which include: pH, DO, pressure, and temperature sensors. All four sensors fall under the category of smart digital sensors, in which serial communication protocol I2C is used for communication between the MCU and the sensors. The sensor module is controlled using TI's MCU TM4C1294XL, which will act as the slave module. The master (controlled by a Raspberry Pi), when necessary, will ping the slave for sensor readings. The MCU will then collect sensor readings and transfer them to the Raspberry Pi through UART.

For proper MCU operation various peripherals were enabled and configured, using the standard peripheral library provided by TI. The system clock was configured to its maximum processing speed, 120Mhz. The peripherals utilized were: three GPIO modules, one I2C module, two timer modules, and one UART module. A more detailed explanation of each peripheral is given ahead.

2.2 I2C Peripheral

I2C is half-duplex communication protocol, meaning it can only transmit or receive data at any given time through its SDA line. Data transfer is synchronized through its SCL line using clock signals. This communication protocol permits multiple slave devices to be connected on the same bus; each having its own unique address.

TI's TM4C1294 microcontroller was properly configured in order to support I2C serial communication protocol. Peripherals, such as the proper GPIO port and I2C port, were successfully enabled. In this case, GPIO port B was enabled, and pins PB2 and PB3 were configured as SCL and SDA, respectively. The I2C port 0 was configured with a transfer rate of 100kbps, in order to accommodate the slowest sensor; which only supports a maximum transfer rate of 200kbps. As per the requirements of the I2C protocol, both the SDA and SCL lines are pulled up to V_{cc} using two 5k ohm resistors.

2.3 UART Peripheral

UART is a full-duplex communication protocol which allows simultaneous data transfer and receiving; this is accomplished by having two separate buses for transferring (Tx) and receiving (Rx). Communication with the master module is achieved using this communication protocol. The MCU contains 8 different UART modules, ranging from zero to seven; module seven was chosen and properly configured. A baud rate of 115200 bps was chosen, as well as the standard 8 data bits with one stop bit, and no parity. In order to send and receive data, GPIO port C was enabled and pins PC4 and PC5 were configured as Rx and Tx, respectfully.

* For proper operation, Tx pin must be connected to Rx pin of other device; and vice-versa.

Table 1: UART settings

UART configuration	
Baud rate	115200
Data bits	8
Stop bits	1
Parity	none

2.4 GPIO

Each peripheral on the custom PCB board is independently powered and have their own dedicated GPIO pins. In order to supply power to these peripherals, GPIO port L was enabled and pins PL0, PL1, PL2, and PL3 were properly configured as output pins. These four pins power, respectively, the Ph, DO, temperature and pressure sensors.

The other GPIO modules have already mean mention and discussed further above.

2.5 Timer Peripheral

Timer utilizes the system clock ticks in order to accurately "keep track" of time. For an MCU operating at 120Mhz, a timer load value of 120M is equal to one second, as it takes 120M ticks for one full clock cycle to be executed (which takes one second). The following formula can be used in order to determine the correct load value for the timers.

$$\frac{x}{System\ clock\ speed} = \frac{time\ needed}{1\ second}$$

All four sensors require a delay after initiating conversion to correctly retrieve data. In order to get an accurate delay for the wait time, two of the six on onboard timer modules were utilized. Both timer0 and timer1 were configured as one-shot; meaning the timers will only countdown and expire once before manually reactivating them. To get the necessary delay of 25ms that's required for the barometer, tiemr-0 was setup with a load value of 240000. On the other hand, to get a delay of 900ms necessary for the Ph sensor, timer-1 was setup with a load value of 108000000.

*For the other two sensors, the appropriate wait time is achieved through a polling method, so no timers are utilized.

2.6 Peripheral and Pin Table

Table 2: Pin Table

Peripheral	Pin	Pin Type	Function
GPIO-B	PB3	SDA	Transmit/receive data (I2C)
	PB2	SCL	Synchronize bit transfer (I2C)
GPIO-C	PC4	Rx	Receive data (UART)
	PC5	Tx	Transmit data (UART)
GPIO-L	PL0	Output	Powers Ph sensor
	PL1	Output	Powers DO sensor
	PL2	Output	Powers Temp sensor
	PL3	Output	Powers barometer
I2C-0	N/A	N/A	Handles communication between MCU and sensors
UART-7	N/A	N/A	Handles communication protocol between MCU and Raspberry Pi

Timer-0	N/A	N/A	Generates delay for barometer
Timer-1	N/A	N/A	Generates delay for Ph sensor

2.7 Sleep Module

The TM4C2194xl MCU offers three different operational modes: run mode, sleep mode, and deep-sleep mode. Each of these different operational modes provides different types of power consumption and wake up times; each useful depending on the functionality needed. Run mode is the "default" operational mode, allowing the MCU to normally operate and execute code, providing no power saving functionality . Alternatively, sleep and deep-sleep mode provide lower power consumption by un-clocking the processor and memory subsystem, preventing the MCU from running code. As the name implies, deep-sleep mode provides lower power consumption when compared to the standard sleep mode. In each of these modes, the user can choose what peripherals remain active, even when the MCU is put into sleep mode. In order to fulfill the sensors hub planned functionality, the only peripheral that will remain active when the MCU enters deep-sleep mode is the UART module. This is done to wake up the device (which can only be done through an interrupt) when the Raspberry Pi sends a command through the UART port. Another difference between the two different types of sleep mode is their respective wake up times. Deep-sleep mode, while greatly reducing power consumption, requires a longer wake-up time compared to the standard sleep mode.

Besides the processor, other system components can also be configured into different power states. These components include: flash memory, LDO, and SRAM. The MCU was configured with the following settings:

Table 1: MCU Operational Modes

Components	Modes	Working Mode
Processor	Run mode	
	Sleep mode	
	Deep-sleep mode	**X**
SRAM	Normal	
	Standby	
	Low Power	**X**
Flash	Normal	
	Low Power	**X**
LDO	1.20	
	1.15	
	1.10	
	1.05	
	1.00	
	0.95	
	0.90	**X**

Based on the above configurations we can estimate the current consumption of the MCU. This estimation can be seen in table 2.

Table 2: I_{DD} Current Consumption

Parameter	Parameter Name	Conditions	System Clock		Nom				Max		Unit
			Frequency	Clock Source	-40°C	25°C	85°C	105°C[c]	85°C	105°C[c]	
$I_{DD\_DEEPSLEEP}$[e]	Deep-Sleep mode (FLASHPM = 0x2)	V_{DD} = 3.3 V V_{DDA} = 3.3 V Peripherals = All ON LDO = 1.2 V	16 MHz	PIOSC	9.74	9.78	10.8	11.6	24.1	32.1	mA
			30 kHz	LFIOSC	2.60	2.83	3.83	4.60	17.1	25.3	mA
		V_{DD} = 3.3 V V_{DDA} = 3.3 V Peripherals = All OFF LDO = 1.2 V	16 MHz	PIOSC	4.53	4.05	4.88	5.53	15.9	22.7	mA
			30 kHz	LFIOSC	0.614	0.762	1.69	2.46	13.3	20.7	mA
		V_{DD} = 3.3 V V_{DDA} = 3.3 V Peripherals = All ON LDO = 0.9 V[f]	16 MHz	PIOSC	5.21	7.33	7.97	8.48	15.3	20.1	mA
			30 kHz	LFIOSC	2.02	2.16	2.79	3.29	10.0	14.9	mA
		V_{DD} = 3.3 V V_{DDA} = 3.3 V Peripherals = All OFF LDO = 0.9 V[f]	16 MHz	PIOSC	1.08	3.10	3.61	4.01	9.50	13.4	mA
			30 kHz	LFIOSC	0.367	0.454	0.954	1.36	6.86	10.8	mA

Table 3: I_{DDA} Current Consumption

Parameter	Parameter Name	Conditions	System Clock		Nom				Max		Unit
			Frequency	Clock Source	-40°C	25°C	85°C	105°C[c]	85°C	105°C[c]	
$I_{DDA\_DEEPSLEEP}$	Deep-Sleep mode (FLASHPM = 0x2)	V_{DD} = 3.3 V V_{DDA} = 3.3 V Peripherals = All ON LDO = 1.2 V	16 MHz	PIOSC	2.45	2.48	2.50	2.48	2.84	2.90	mA
			30 kHz	LFIOSC	2.45	2.48	2.50	2.48	2.85	2.90	mA
		V_{DD} = 3.3 V V_{DDA} = 3.3 V Peripherals = All OFF LDO = 1.2 V	16 MHz	PIOSC	0.226	0.227	0.265	0.249	0.558	0.635	mA
			30 kHz	LFIOSC	0.228	0.227	0.272	0.247	0.558	0.600	mA
		V_{DD} = 3.3 V V_{DDA} = 3.3 V Peripherals = All ON LDO = 0.9 V[f]	16 MHz	PIOSC	2.14	2.42	2.44	2.42	2.78	2.88	mA
			30 kHz	LFIOSC	2.44	2.42	2.44	2.42	2.86	2.88	mA
		V_{DD} = 3.3 V V_{DDA} = 3.3 V Peripherals = All OFF LDO = 0.9 V[f]	16 MHz	PIOSC	0.216	0.166	0.209	0.193	0.563	0.580	mA
			30 kHz	LFIOSC	0.223	0.167	0.209	0.189	0.508	0.580	mA

The total current consumed is the addition of I_{DDA} and I_{DD}. Assuming all peripherals are off and an LDO volage 0.9 (as we have configured), at clock frequency of 16MHz (PIOSC), taking 25 degrees Celsius to be the operational temperate, the total current consumed in deep-sleep mode is:

$$I_{total} = 0.454mA + 0.167mA = \ 621\mu A$$

As seen in table 1, all components have been put in their least power consumption modes, depending on the modes selected the MCU's wake up time can either increase or decrease. Using the current configuration, we can approximate the wake-up time to be.

Table 4: Deep-Sleep Wake Up Times

Parameter No	Parameter	Parameter Name	Min	Nom	Max	Unit
D8	$T_{PIOSCDS}$	Time to restore PIOSC as System Clock in Deep Sleep Mode	-	-	14	Deep Sleep Clock Cycles[a]
D9	T_{MOSCDS}	Time to restore MOSC as System Clock in Deep Sleep Mode	-	-	18	ms
D10	T_{PLLDS}	Time to restore PLL as System Clock in Deep Sleep Mode	-	-	1 cycle of Deep Sleep Clock + 512 cycles of PLL reference Clock[a]	clocks
D11	T_{LDODS}	Time to restore LDO to 1.2 V in Deep Sleep Mode	-	-	39	µs
D12	$T_{FLASHLPDS}$	Time to restore Flash to active state from low power state	-	-	5	µs
D13	$T_{SRAMLPDS}$	Time to restore SRAM to active state from low power state	-	-	15	µs

Table 4 shows the different wake up times for the processor and various components of the MCU. The wake-up time for the MCU is calculated as following:

$$Wake\ Time = T_{PIOSCDS} + T_{PLLDS} + T_{SRAMLPDS}$$

2.8 Wiring Diagram

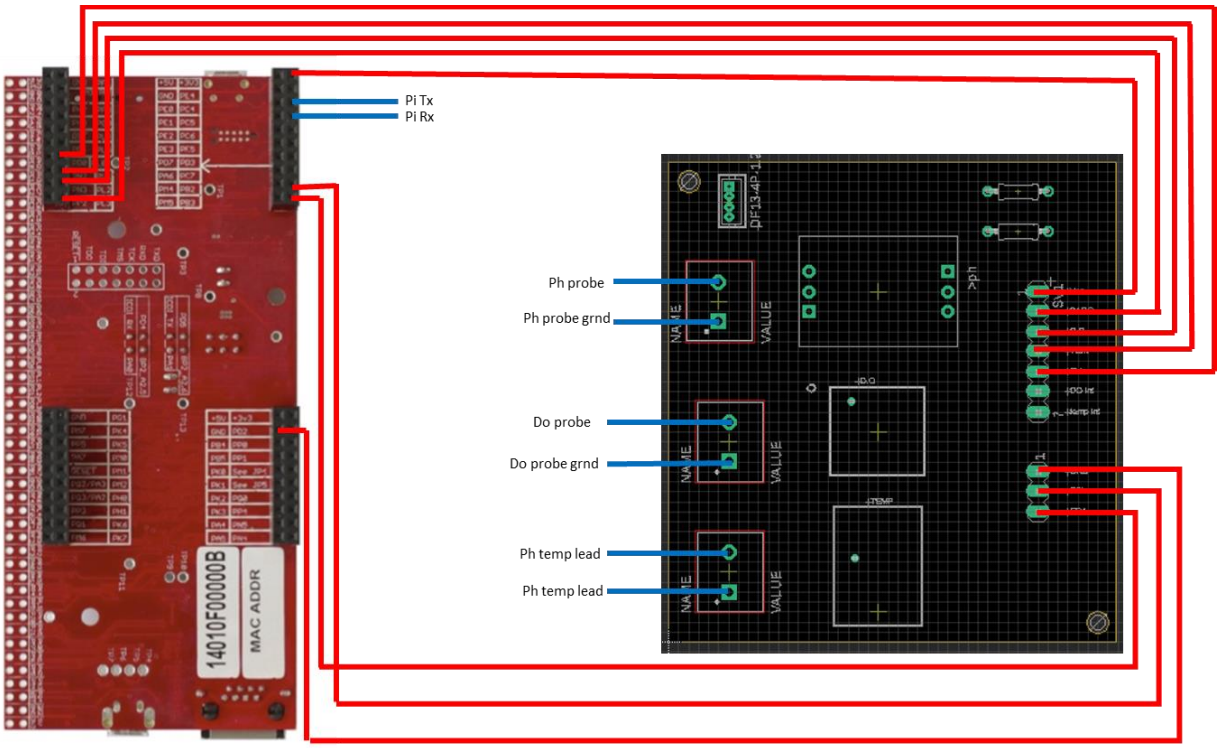


Figure 1 Wiring Diagram

3.0 Single sensors

3.1 Temperature Sensor

Temperature sensor communication was first established using the TSYS01 sensor. The approach used, which was later implemented in code, can be seen in figure 2, which can be found in the device's data sheet.

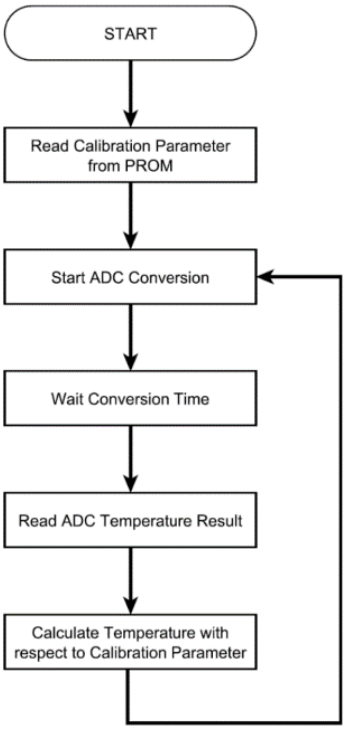


Figure 2 TSYS01 Reading Procedure

It was later decided that instead of using this particular sensor, the temperature sensing capabilities of the PH probe (both the DO and PH probes come with temperature sensing capabilities and can act as temperature sensors). In order to access the temperature sensing capabilities of the probes, an RTD-4-OEM circuit is needed, which performs the necessary temperature conversions and allow us to communicate to it via the I2C serial communication protocol. In order to get readings from the device the following approach, which can be seen in figure 3, was used.

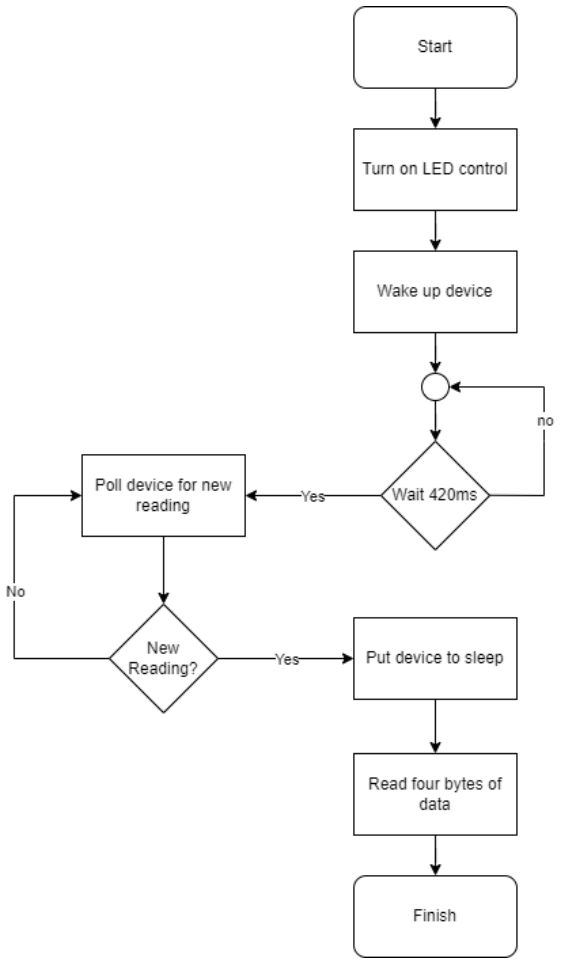


Figure 3 RTD-4-OEM Reading Process

In order to take a reading, the device must be woken up from its default sleep state. Once the device is woken up, it will continuously take readings every 420ms, overriding the last reading. As result, once 420ms have passed and a reading has been taken, the device must be put to sleep before reading its contents. The led indicator is turned on at the beginning in order to determine when a reading ha taken place.

3.2 Pressure

In order to measure pressure, the MS5837-30BA pressure sensor was used from Blue Robotics. This particular sensor is able to measure up to 300m in depth and offers temperature compensation capabilities in order to give more accurate readings. A similar approach to that of the TSYS01 temperature was used in order to achieve communication and extract data from the device.

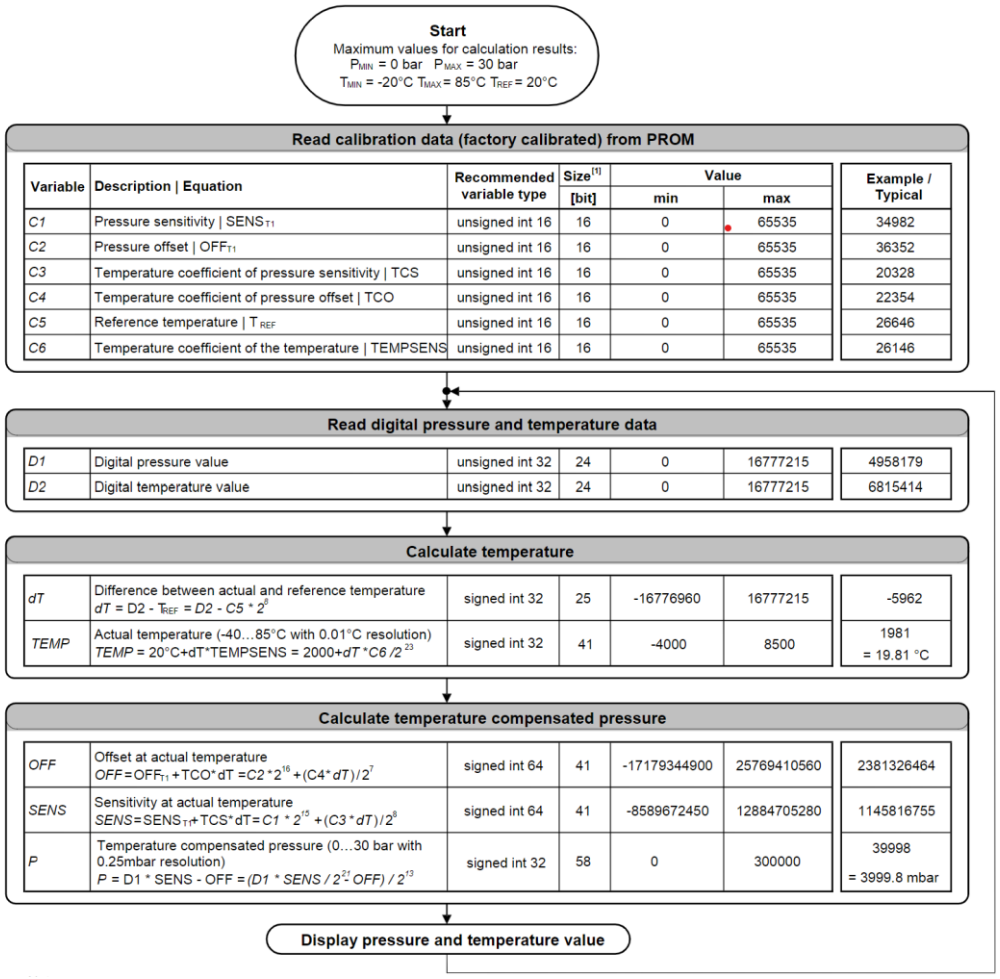


Figure 4 MS5837-30BA Reading Procedure

As per the data sheet, four calibration coefficients must be read first in order to perform the necessary calculations. After obtaining the necessary pressure conversion and temperature conversion from the device, the equation appearing in figure 4 most be implemented, using the calibration coefficients and pressure conversions, in order to obtain the correct pressure value.

3.3 PH

The ENV-50-pH industrial probe, from Atlas Scientific, was used to measure pH levels. In order to extract readings from the industrial probe, an additional circuit is needed, the ENZO-pH embedded circuit. This embedded circuit performs the calculations and conversions necessary to give an accurate pH reading. Furthermore, it allows us to extract that from the device using I2C serial communication protocol. The embedded circuit gives and receives data in ASCII format. In figure 4, the approach used in order to extract one reading from the sensor can be seen.

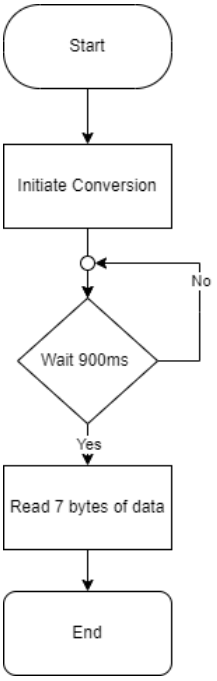


Figure 5 ENV-50-pH Reading Procedure

The devices onboard led will flash green when taking a reading, and flash red when an unrecognizable command has been sent. As seen in figure 5, once the data conversion has been initialized, one must wait 900ms in order to read data. If a read command is issued before the necessary time has elapsed, no useful data will be received. On a separate note, this probe also comes with temperature sensing capabilities, but in order to extract temperature readings a different embedded circuit is needed.

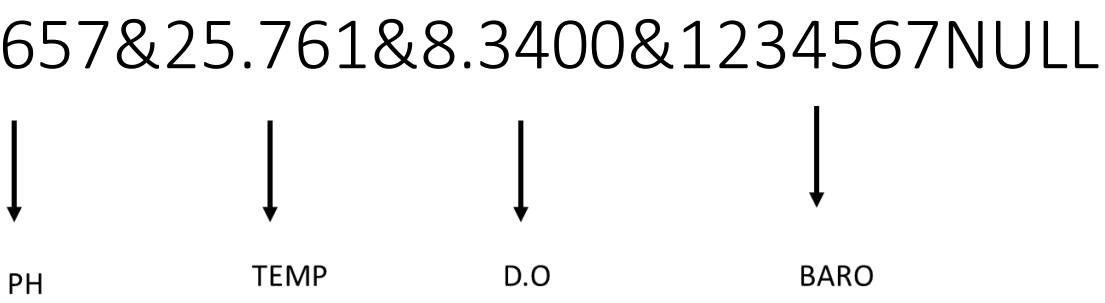
3.4 DO

Dissolved oxygen levels in the water are given by the ENV-50-DO industrial probe, from Atlas Scientific. Similar to the pH industrial probe mentioned above, an additional embedded circuit is needed in order to request a reading from the sensor. The DO-OEM circuit was used to achieve this purpose. Like the RTD-4-OEM and the ENZO-pH embedded circuit, the DO-OEM circuit performs the necessary conversions for us, giving us accurate values. Like the other two embedded circuit mentioned above, this device gives data in ASCII format. The approach used to extract data from the device can be seen in figure 3, as they both work the same and require the same procedure in order to fetch readings.

4. System Operation

At first, when the system is first powered on the MCU will setup and configure its necessary peripherals, as discussed above. Furthermore, at startup, the MCU does a system check, where it “pings’ the sensors to determine their operational state; the working principal behind the function is described in greater detail in the next section. After performing these two tasks the MCU goes into deep-sleep mode, waiting for a command from the raspberry Pi. Once the MCU receives a command through UART, it will wake up and perform one of its already defined set of instructions; if by any chance the MCU doesn’t recognize the command it will go back to deep-sleep mode.

Ideally, when the MCU is instructed to fetch readings, conversion from all four sensors would be initiated simultaneals, resulting in less overall wait time. However, because of the chosen method, this is not possible. Temperature is a factor that affects both dissolved oxygen levels, as well as Ph levels. The MCU first takes a pressure reading, and then a temperature reading. The MCU then passes the temperature value to both the Ph and DO sensors, in order for them to do their internal temperature compensation; for most accurate readings. Once the temperature value has been loaded, sensor readings are requested from both sensors, almost simultaneously. Firstly, data is fetched from the fastest sensor, as to lower the overall wait time. Once the 420ms wait time has elapsed data is fetched from the DO sensor. Finally once the 900ms wait time has elapsed, data is fetched from the Ph sensor. The MCU then sends the data back to Raspberry Pi an goes back to sleep, until woken up again. The data package sent can be seen ahead.



The first five bytes belongs to the pH sensor, the next six bytes belong to the temperature sensor, the next six bytes belong to the dissolved oxygen sensor, and lastly the last seven bytes belong to the barometer. Each sensor reading is separated by a “&” character. The string is terminated with a NULL character, AKA 0x00

4.1 System Check

As mentioned above, on startup, the MCU “pings” the sensors to determine their operational condition. The MCU requests a reading from the sensors and compares the first data bit from each transmission to a specific predefine value. If the first bit matches the predefined value, it assumes the sensor is in working condition and moves on to the next sensor. Consequently, if the values do not match the sensor is turned off and on to trigger a power on reset, and the process is repeated once more before moving on with the next sensor. For both the dissolved oxygen and temperature sensors, when a new reading is available, they return a “1”. Similarly, the first bit of the data transmission is also one. Lastly, the barometer returns a NULL character when data is fetched before waiting for the appropriate delay time. These are the predetermined values which the comparison is based on.

The working state of each sensor is recorded in a buffer by writing a “1” or “0” to the corresponding position. If “1” is written it means the sensor passed the system check and is functional, while a “0” represents the opposite. If the sensor is determined to not be functional, it must remain powered on, if any of the sensors are powered off, for unknown reasons, it will prevent proper I2C bus communication.

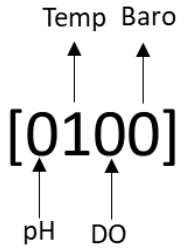


Figure 6 State Data Package

As can be seen above, the first bit represents the pH sensor, the second sensor represents the temp sensor, the third bit represents the DO sensor, and finally the fourth bit represents the Barometer. Looking at figure 1, only the Temperature sensor is in working condition. Once this buffer is sent to the Raspberry Pi, it is its responsibility to properly process the package and apply the corresponding functionality (if any).

4.2 Command Table

Table 2: Command Table

Command	Description
Read	Fetch sensor readings
PhCal	Get Ph readings every second
PhMidCal	Perform mid-point calibration
PhLowCal	Perform low-point calibration
PhHighCal	Perform high-point calibration
DoCal	Ger Do readings every second
DoAtmoCal	Calibrate to atmospheric oxygen content
DoZeroCal	Calibrate to zero oxygen content
TempCal	Fetches temperature reading every second
TempNewCal	Calibrates sensor using a specific temp value
exit	Manually exit form calibration process

5.0 Calibration

5.1 pH Calibration

The pH probe required three different levels of calibration: high point, mid-point and low point calibration; each requiring their own different solution. A solution containing a pH level of 10 was used for high point calibration, a pH level of 7 was used for mid-point calibration, and a pH level of 4 is used for low point calibration. In order to calibrate the probe accordingly, the probe must be submerged in one of these solutions. Once the reading slowly stabilizes and the correct pH reading is read, a command is sent to the device to calibrate the probe. The same process must be followed for the remaining two calibration points. By doing all three calibration points the sensor will provide high accuracy over the full pH range.

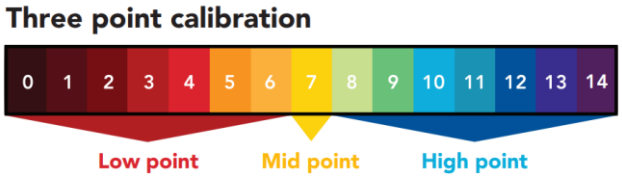


Figure 7 Calibration Range

As mentioned above, the user must see and wait for the readings to stabilize before performing the proper calibration. To request continuous reading from the MCU, the command “PhCal” is used. The before mentioned command takes and sends a reading every second. Once the user is ready to proceed with the calibration, “PhMidCal”, “PhLowCal”, and “PhHighCal” commands are used for md-point, low-point, and high-point calibration; respectively. Once all three-point calibrations are called, the MCU will automatically stop taking reassigns and go to sleep. If the user wants to end the calibration process without sending all three commands, the user can exit by utilizing the “exit” command.

*It’s important to note that once the mid-point command is sent, the remaining two calibration points are automatically cleared and must be recalibrated.

*Probe has already been calibrated

5.2 DO Calibration

Similarly, to the pH sensor, the dissolved oxygen sensor provides two different calibrations; atmospheric oxygen and zero oxygen calibrations. To calibrate the probe to the atmospheric oxygen content, the probe must be exposed to air until the readings stabilize. Likewise, to calibrate the probe to a zero-oxygen content, the probe must be submerged in a special solution till the readings stabilize. This last calibration is only required for accurate readings below 1.0mg/L.

As mentioned above, the user must see and wait for the readings to stabilize before performing the proper calibration. To request continuous reading from the MCU, the command “DoCal” is used. The before mentioned command takes and sends a reading every second. Once the user is ready to proceed with the calibration, “DoAtmoCal” and “DoZeroCal” commands are used for atmospheric oxygen, and zero-oxygen ; respectively. Once both calibrations are called, the MCU will automatically stop taking readings and go to sleep. If the user wants to end the calibration process without sending both commands, the user can exit by utilizing the “exit” command.

*Probe has already been calibrated

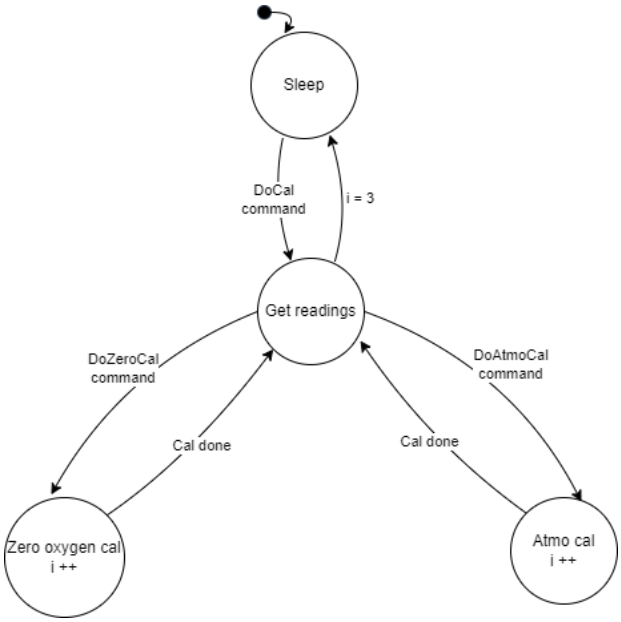


Figure 8 DO calibration State Diagram

5.3 Temp Calibration

Unlike the other two sensors, the temperature sensor only has one type of calibration. The Raspberry Pi must send the temperature value of the solution being measured. To request continuous readings, the command “TempCal” is used. Once the readings have stabilized, and the user has supplied the temperature value, the command “TempNewCal” is used to calibrate the sensor.

*Probe has already been calibrated

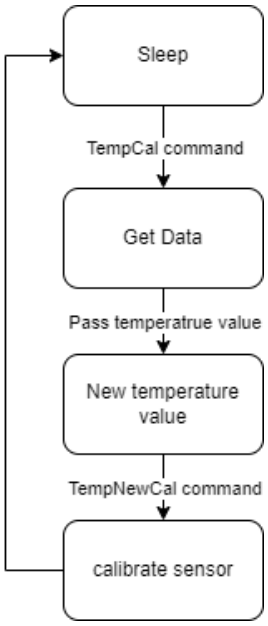


Figure 9 Temperature Calibration

6 Thermal/Power Analysis

Table 3: Power Analysis

Device	Voltage (V)	Current (mA)
pH (ENZO-pH)	3.3	14
Pressure (MS5837-30BA)	3.3	1.20
Temperature (RTD-4-OEM)	3.3	4.2
DO (DO-OEM)	3.3	12
TM4C1294XL	3.3	107.2
Total	-	138.6
Total Power	-	457.38 Watts

In order to determine if the TM4C1294 microcontroller could handle the power dissipation under the full load of the sensors, a thermal analysis was carried out. The thermal analysis consists in calculating the MCU's junction temperature and comparing it to the maximum junction temperature it can handle. This was done by using the following formula:

$$T_j = T_A + \left(\theta_{ja} * P_{diss} \right)$$

Where P_{diss} can be calculated the following way:

$$P_{diss} = V_{dd}(I_{dd} + \sum I_{io})$$

After conducting the thermal analysis, we determined that the MCU could handle the full load that it would be subjected to; allowing the sensors to be connected directly into the GPIO port of the MCU. On the other hand, if the MCU could not handle the full load, the strategy of power gating would have been implemented. This consists of feeding the load using an external power source and using transistors to act as switch, while using the MCU's to control their state.

7 PCB

With the aim to interface all the different sensors under one I2C bus, a custom PCB board was designed and ordered. A board was already previously manufactured, but with the recent change of temperature sensors, a new board needed to be designed. The DO-OEM, RTD-4-OEM and the ENZO-pH embedded circuits, as well as the MS5837-30BA pressure sensor will all be mounted on the board. To achieve this, the PCB board design program, Eagle Cad, was utilized. Firstly, a custom library device package was made for each of the embedded circuits, this include the foot print used in order to design the

board, as well as the symbol used to make the schematic. Before designing the board, a schematic was made in which all the connections between the different components were set up appropriately. As can be seen in the figure 7, the board consists of:

- 2x Screw Terminal Block
- 1x 7 Pin Male Header
- 1x 3 Pin Male Header
- 2x 5 Ohm Resistor
- 1x DF13 Connector
- ENZO-pH
- MS5837-30BA
- DO-OEM

.

Once the schematic was done and the connections were revised, the PCB board was then made. The necessary files used for manufacturing were generated and ordered.

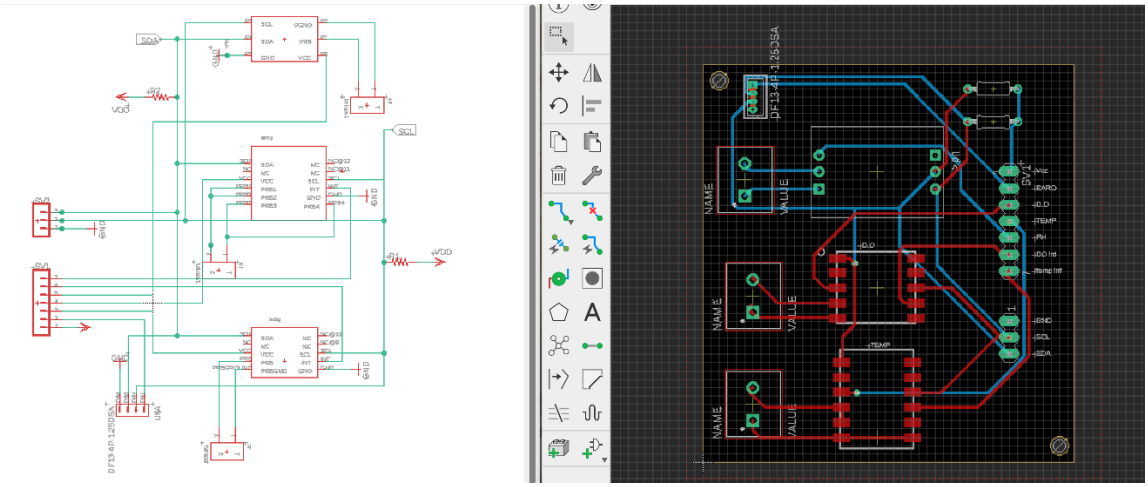


Figure 10 PCB Board and Schematic

8 Raspberry Pi

8.1 Raspberry Pi Considerations

Connection between the Raspberry Pi and the sensor hub is straight forward, only few considerations should be taken when writing the appropriate code; in regards to the sensor hub. When sending any command to the sensor hub the first character sent must always be 0x0D, better known as ”\r”.

GPIO pins 14 and 15 are used on the Raspberry Pi for UART communication.

8.2 Raspberry Pi Proposed Calibration Window

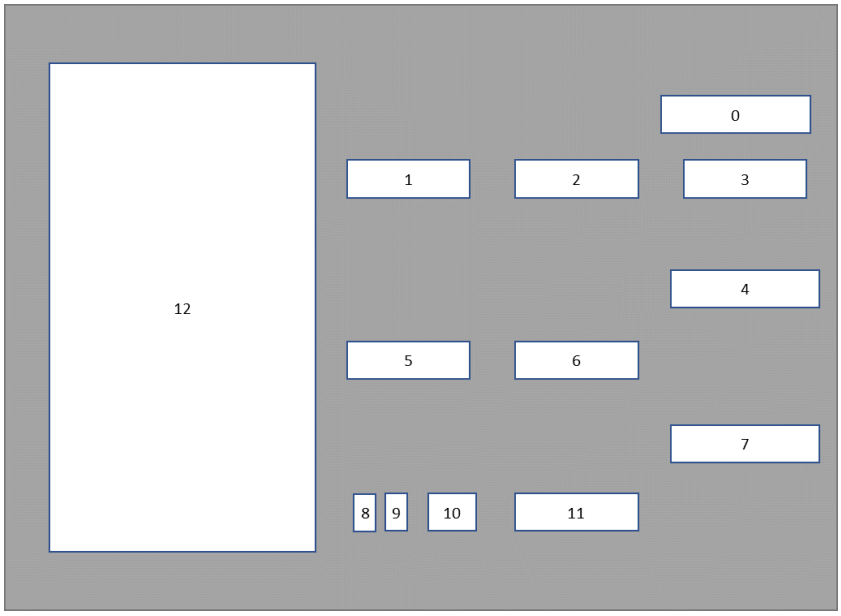


Figure 11 Proposed Calibraiton Window

0: “PhCal” command. This command will send readings every second

1: “PhMidCal” command. This command does the mid-point calibration.

2: “PhLowCal” command. This command does the low-point calibration.

3: “PhHighCal” command. This command does the high-point calibration.

4: “DoCal” command. This command will send readings every second.

5: “DoAtmoCal” command. This command does the atmosphere oxygen content calibration.

6: “DoZeroCal” command. This command does the zero-oxygen content calibration.

7: “TempCal” command. This command will send readings every second.

8-10: placement of new temperature value to be calibrated.

11: “TempNewCal” command. This command does the temperature calibration to the provided value

12: terminal where to display life sensor readings.

9 Work Done/Future Works

Things to do to finish sensor hub :

- Barometer Integration

Although the sensor hub is almost fully finished, in order to utilize it and acquire sensor readings the following must be completed first:

- Accompanying Raspberry Code, to communicate with MCU and receive sensor data.

10 Conclusion

M.O.S.I.S intends to facilitate the study of aquatic organisms, creating a better and more effective way to conduct underwater research on sea specimens. This project aims to remove the need to transport the organism from its habitat, by essentially becoming a lab, replacing other conventional methods of study. To achieve this, M.O.S.I.S will be able to capture micro/stereoscopic images, along side environmental data. During the academic year of 2022-2023, all efforts have been exclusively put into the attempt to finish and implement the sensor hub; all work done till now has been documented in this report. The sensor hub houses four different sensors: a temperature, pressure, DO, and pH sensor; all supporting I2C serial communication. Work done till now can be summarized into two classifications: MCU configuration for I2C communication, and code development for the extraction of data from individual sensors. As of this point, communication with the four sensors has been successfully established, MCU has been properly configured and initialized, sleep functionality has been configured, PCB board has been designed, system diagnostic and error-handling routines have been developed, sensors have been calibrated, and a means for data extraction has been developed. The overall project objective is to determine effectiveness of MOSIS prototype vs other conventional methods. In order to accomplish this, the effectiveness of this prototype will be evaluated by applying a study of diseases using two methods: In-situ observations and sample extraction, to be observed in a laboratory. Observations of diseases such as SCTLD, Coral Bleaching, and other coralline border phenomena.

References

[1] TSYS01-FAMILY Digital Temperature Sensors, TE Connectivity Company, Phillipsburg, NJ, 2017.

[2] MS5837-30BA Ultra-Small, Gel-Filled, Pressure Sensor with Stainless Steel Cap, TE Connectivity Company, Phillipsburg, NJ, 2017.

[3] EZO-pH™Embedded pH Circuit v5.9, Atlas Scientific Environmental Robotics, Jacksonville, Florida, 10/21.

[4] OEM-DO™Embedded Dissolved Oxygen Circuit v3.3, Atlas scientific Environmental Robotics, Jacksonville, Florida, 10/21.

[5] OEM-RTD™4 Wire – Embedded Temperature Circuit v1.8, Atlas scientific Environmental Robotics, Jacksonville, Florida, 10/21.

[6] Saavedra, S., 2020. *MOSIS Research Final Report*. pp.1-42.